AE 353 Project 2: Differential Drive Robot

Anshuk Chigullapalli[1]
University of Illinois at Urbana-Champaign, Champaign, Illinois, 61820, USA

The goal of this project is to create a feedback controller that actuates the two wheels of a differential drive robot (known as "segbot") to keep it upright and move it along a given circular track. The report outlines the additional benefits and drawbacks of the designed controller, compares a few tests of the controller, and describes a few modifications to the simulation. The controller and the simulation are designed, implemented, and tested in a Jupyter Notebook python environment.

I. Nomenclature

e_l	=	Lateral Error of the robot
e_h	=	Heading Error of the robot
f	=	The equations of motion as a function of state and inputs
τ_r	=	Torque applied to the right wheel
τ_l	=	Torque applied to the left wheel
θ	=	Pitch angle of the robot
$\dot{\theta}$	=	Pitch rate of the robot
u	=	The vector of inputs of the system
v	=	Forward velocity of the robot
w	=	Turning rate of the robot
x	=	State vector

II. Introduction

Different drive robots are robots which have wheels that are supplied independent torques, rather than having an axle system like in conventional automobiles. This independent motor mechanism is often used on small to medium sized robots to improve their mobility and maneuverability. One of the best examples of a differential drive motor, and one that is extremely similar to the model we will be using for this project, is the "Segway", a self-balancing personal transporter used for short distance travel[2].

Fig. 1: A Segway Ninebot S

The model for this project is also very similar: a self-balancing device that can be directed in certain directions. The figure below shows the original model provided for the project.

[1] Student, Department of Aerospace Engineering.
[2] https://en.wikipedia.org/wiki/Segway

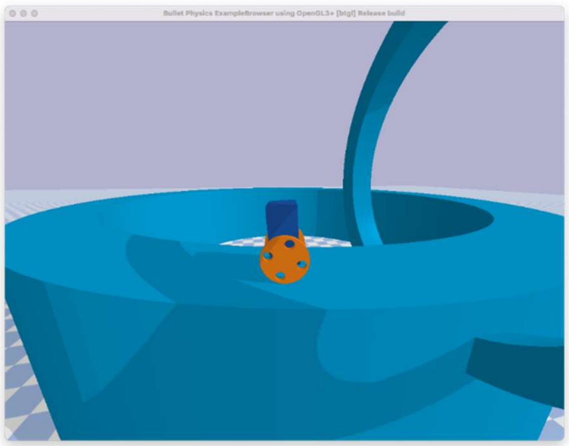
Bullet Physics ExampleBrowser using OpenGL3+ [btgl] Release build

Fig. 2: Original Segbot

The sections of this report go over the requirements set for this project, the design process involved in developing a feedback controller for this system and the testing to verify that the system achieved all the set goals.

III.Requirements and Validation

The efficacy of our system is dependent on the requirements we choose for the system. If our requirements are met, then the system is successful. If not, then it requires further improvement. The following statement is the requirement set for this design project. This is by no means the most difficult requirement, but it still requires a properly working tuned controller. In the future, this requirement can be made more challenging and the controller can be improved further.

Requirement: The robot must be capable of completing more than 5 laps of the given circular flat circuit without falling over, with the lateral displacement e_l within $\pm$ 0.3 m at any point in its journey, and no lap taking greater than 25 seconds. If the initial condition sets the robot at $0.6 > |e_l| > 0.3$, then the robot must return to $|e_l| < 0.3$ within the first 2.5 seconds. No requirement is set for when $|e_l| > 0.6$. All other initial conditions are set to 0.

The requirement that the robot must complete more than 5 laps is to ensure that the controller is consistent, rather than saying the controller is good because it satisfies the requirements for a single lap. In the design of real controllers, this consistency is far more valuable than super powerful yet unreliable controllers.

These requirements have to be tested and validated. For this, we set up the following verification procedure:

- Pybullet will be used to simulate the system.
- The measurements will be extracted from this system and analyzed. Specifically, we will analyze the lateral displacement (perpendicular distance between the wheel center and the track center) and ensure that it remains under 0.3 m.
- We will also observe the pitch angle θ, the angular deviation of the chassis from its perfectly upright position, to ensure that the robot didn't fall over. This will also be monitored using Pybullet's graphical simulation window.
- The motion of the robot will be monitored in the graphical window to ensure that the robot completes 5 laps of the circular track.
- An external stopwatch will be used to find the lap time of each lap. For future purposes, the simulation code can be used to track position and print the times when the robot crosses the start line, but for now it will be done using an external device and monitoring the graphical simulation window.

With the requirements and verification method set, the controller can now be designed and tested.

IV. Pre-LQR setup

A. Equations of Motion and Linearization

1. Picking State Variables

We can first pick the state vector of the system x. Based on the variables that the simulation captures, we can set the state of the system to be defined as follows:

$$x = \begin{bmatrix} e_l \\ e_h \\ v \\ w \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{1}$$

We have two inputs in our system, the torque supplied to the left wheel and the torque supplied to the right wheel. These are set in the input matrix u.

$$u = \begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix} \tag{2}$$

The equations of motion for this system were provided with the design problem. We can put the equations of motion in the form of $\dot{x} = f$, where f is the matrix with all the equations of motion, shown below.

$$f = \begin{bmatrix} v\sin(e_h) \\ w \\ -\dfrac{2400\tau_L+2400\tau_R+2808\left(\dot{\theta}^2+w^2\right)\sin(\theta)+13\left(250\tau_L+250\tau_R-195w^2\sin(2\theta)-8829\sin(\theta)\right)\cos(\theta)}{11700\cos^2(\theta)-12168} \\ \dfrac{32\left(-875\tau_L+875\tau_R-1443\dot{\theta}w\sin(2\theta)-2925vw\sin(\theta)\right)}{13\left(3120\sin^2(\theta)+2051\right)} \\ \dot{\theta} \\ \dfrac{42250\tau_L+42250\tau_R-32955w^2\sin(2\theta)+300\left(100\tau_L+100\tau_R+117\left(\dot{\theta}^2+w^2\right)\sin(\theta)\right)\cos(\theta)-1492101\sin(\theta)}{1404\left(25\cos^2(\theta)-26\right)} \end{bmatrix} \tag{3}$$

Just the equations of motion are not enough to linearize the system. We also we need to set an equilibrium point for it to be linearized around. The equilibrium point is set with the following values:

$$x_e = \begin{bmatrix} 0 \\ 0 \\ 4 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4}$$

This works as an equilibrium point because all the elements of f evaluate to zero at this point. This equilibrium point was set with $v_e = 4\frac{m}{s}$ because the system will for the most part be moving around this velocity and not be stationary. We want most of the other values to converge or stay small (turning rate comes in this second case) and that's why they are all 0.

2. Calculating A and B

The Jacobian of f is calculated with respect to the state and input matrices to create functions for A and B. These A and B functions are then evaluated at the equilibrium point. The result is an A and B that represent the system linearized at that equilibrium point. For our system and the equilibrium point described before, we get the following A and B. This whole process was implemented in the python notebook.

$$A = \begin{bmatrix} 0 & 4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -245.25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1062.75 & 0 \end{bmatrix} \tag{5}$$

$$B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 12.0726 & 12.0726 \\ -1.050 & 1.050 \\ 0 & 0 \\ -51.4601 & -51.4601 \end{bmatrix} \tag{6}$$

We now have a linearized system of the state space form shown below.

$$\dot{x} = Ax + Bu \tag{7}$$

B. Checking for Controllability

Not all state space models are controllable, i.e. no matter what feedback loop is designed, an uncontrollable system would never converge to the desired state. This is why we need to check whether our system is controllable before we design our controller.

To do so, we find the controllability matrix W. It is defined as shown below:

$$W = [B\ AB\ A^2B\ A^3B\ \ldots A^{n-1}B] \tag{8}$$

In the above expression, n is equal to the number of states. In this case, $n = 6$. The controllability matrix is calculated in the python notebook. The rank of the controllability matrix is what determines whether the system is controllable or not. If the rank is equal to the number of states in the system, then the system is controllable.

The rank was found using the numpy.linalg.matrix_rank() function. We see that W has a rank of 6, which means that our system, for our choice of states, is controllable.

V. LQR controller design and tuning

Now the goal is to design a feedback controller with controller gain matrix K, where:

$$u = -Kx \tag{9}$$

In the previous design project, we tested random values of K until we found a set that gave a stable response that also met our requirements. This time, we calculate the Linear Quadratic Regulator controller, which is optimized to minimize a given cost function. The cost function is set by diagonal matrices Q and R, which can be modified to change the amount of error and effort we want our system to be penalized for. This is a much more intuitive system that lets us tune the controller for better results far more easily that looking for random K values or placing eigenvalues manually. In the notebook, LQR is implemented by solving the continuous absolute Ricatti equation to get P. Using that value, we can calculate K.

$$K = R^{-1}B^TP \tag{10}$$

The main task now becomes finding the values to place in Q and R. Listed below are some of the basic thoughts that were the starting point for the tuning process:

- The lateral error and heading error need to be strongly penalized as larger values of lateral or heading error could cause the robot to fall off the track,
- The difference in velocity and the desired velocity doesn't need to be penalized too much, as reaching the desired velocity isn't as important as keeping the robot on track and upright.
- The turning rate error is penalized moderately, as it is the derivative of the heading error.

- The pitch and the pitch rate errors are highly penalized because a large pitch could have extremely bad consequences for our robot (i.e. falling over)
- R is set to be an identity matrix, as any change in its values (which would have to be equal to each to each other) could just be implemented by scaling down values in Q.

With these general ideas in mind, many different values of Q can be found. This is where thoroughly tuning and testing the system becomes very important.

A. Modifications to the Simulation

For purposes of making testing easier, the simulation environment was modified to not handle one robot but two, with each robot having its own sensor measurements and controller. This way, if one robot performs better than the other, then that controller is retained, and the other controller is tuned further to possibly get a better result! This iterative process is continued until a satisfactorily "good" controller is created. I found this particularly useful because it was far easier for me to compare two controllers when I could see them travelling on the same path together, and For more information on what determines "good", refer to Section III, and for the testing system refer to Section VI.

Most of the modifications make to the simulation were in the internal setup code and the 3D model files. Some of the changes in the notebook include a Boolean that can be flipped to switch from a single robot simulation to a dual robot simulation. For more information on the exact code modifications, refer to the submitted code.

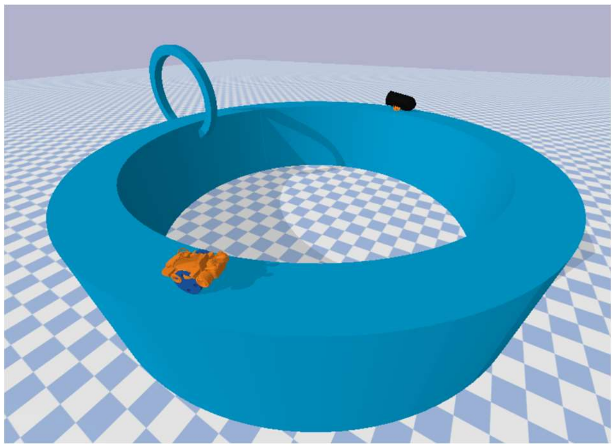

Figure 3: Simulation with Mario Kart Models

The models of the two robots are based on a standard kart and the Bullet Bill from the Mario Kart video game franchise[3]. The Bullet Bill was generally assigned the better controller, and the standard kart was given the controller that need to be tested. If the controller under testing turned out to be better, then the bullet bill would be given that controller.

B. Final LQR results

After rigorous tuning of the controller by modifying the Q and R values, the final chosen values were as follows:

$$Q = \begin{bmatrix} 50 & 0 & 0 & 0 & 0 & 0 \\ 0 & 35 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 \\ 0 & 0 & 0 & 0 & 60 & 0 \\ 0 & 0 & 0 & 0 & 0 & 40 \end{bmatrix} \tag{11}$$

$$R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{12}$$

[3] The models for the Bullet Bill and the kart were not made by me. They were downloaded from GrabCAD (an online 3D file sharing website). The links for the models are provided: https://grabcad.com/library/yoshi-mario-kart-1 https://grabcad.com/library/bill-ball-team-1

With the cost function created by these two matrices, we use the system described at the start of Section V to find the value of K. Given below is that calculated value.

$$K = \begin{bmatrix} -5.000 & -14.596 & -0.707 & -4.888 & -23.119 & -4.691 \\ 5.000 & 14.596 & -0.707 & 4.888 & -23.119 & -4.691 \end{bmatrix} \tag{13}$$

We will now elaborate on the results of this controller and see if it satisfies all of our problem requirements.

VI. Results

The above controller was implemented in the pybullet simulation environment. The parameters for this simulation are set as follows:

- The simulation only uses a single robot.
- $v_{des} = 4\frac{m}{s}$. This is a fast-enough speed for the robot to complete a single lap within 25 seconds.
- $e_{l_{des}} = -0.3$. During testing it was seen that there was some positive steady state error in the lateral error that didn't seem to diminish even with large derivative costs. Therefore, I decided to set a negative desired value to cancel out the steady state error.
  Note: If the turn_left condition is set to true, then this desired lateral error needs to become 0.3.
- All the other desired values are set to 0.
- All the initial conditions are set to 0, and the turn_left condition is set to False.
- The simulation time is set to 125 seconds. If the robot cannot finish 5 laps in that time frame, then it does not satisfy the requirements.

The plots below show the result of this test.

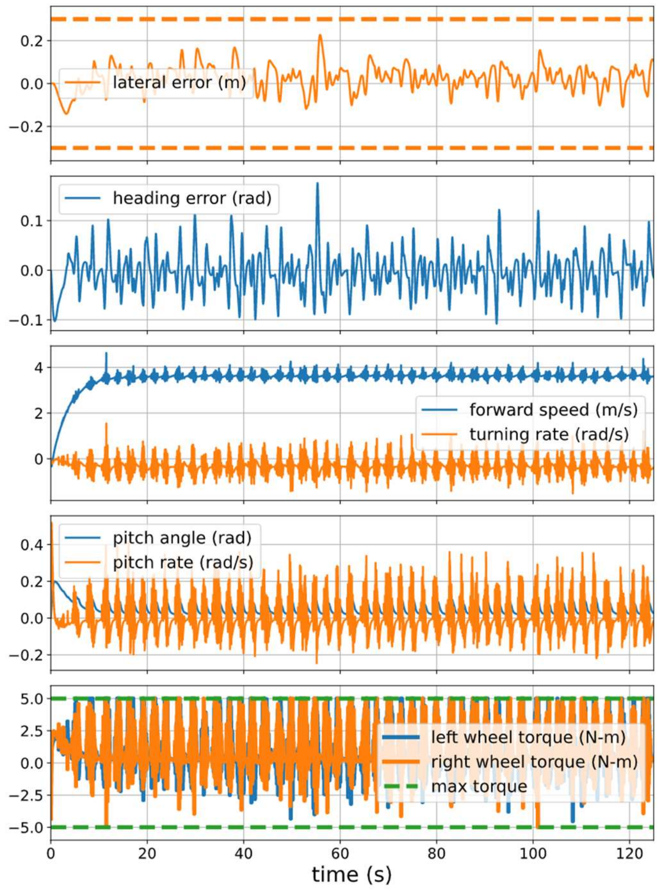

Figure 4: Results of the 125 second simulation

We see in these results that the lateral error remains under 0.3 m for the entire duration, easily satisfying our requirement. We also see that the pitch angle fluctuates between 0 and 0.1 radians, but it doesn't diverge to large values that result in the robot falling over. The video submitted along with this report contains a sped-up version of this 125 second simulation to confirm that the robot does indeed complete 5 laps, with no lap taking longer than 25 seconds.

Below is a table that lists the amount of time each lap took. The video shows a stopwatch for proof.

Table 1: Lap times

Lap	Time (s)
1	21.54
2	16.01
3	17.64
4	17.93
5	17.55
Total:	90.67

The last part of the requirement was that if the initial lateral error > 0.3, then that measurement should go down within the first 2.5 seconds. The plot below shows the lateral error from a simulation of a single lap, with the initial $e_l = 0.5\ m$.

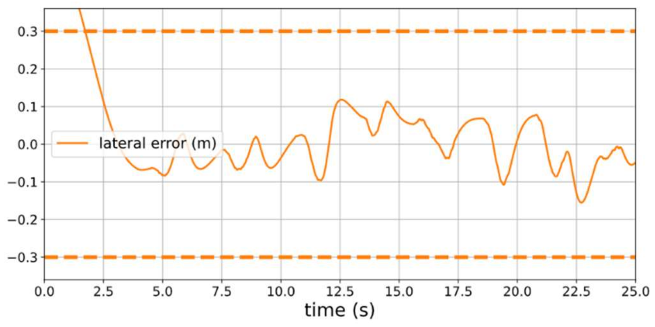

Figure 5: Large Initial Lateral Error Test

We see that the system seems to get below 0.3 m of lateral error before the 2.5 second mark.

With these conclusions and the other data shown visually in the video, we can confirm that all the requirements (lateral error, lap time, not falling over) are all met.

One important aspect to note in this is the fluctuations in a lot of the sensor measurements. This is most probably a residue of our equations of motion being for a straight line, and not for a circular track. In the real simulation, we do need a small amount of turning rate, and so the torques act accordingly.

VII. Different Desired Velocities

The requirements set for this controller design were all met in the results of the previous section, but for the sake of exploring further, the controller can be tested for different desired velocities to see how the behavior of the system changes.

The results in the previous section were with a desired velocity of $4\frac{m}{s}$. In this section, we will look at the results of having the desired velocity at $5\frac{m}{s}$ and $2\frac{m}{s}$

The following plot shows the sensor data collected from a simulation run for 25 seconds where the desired forward speed is set to $5\frac{m}{s}$.

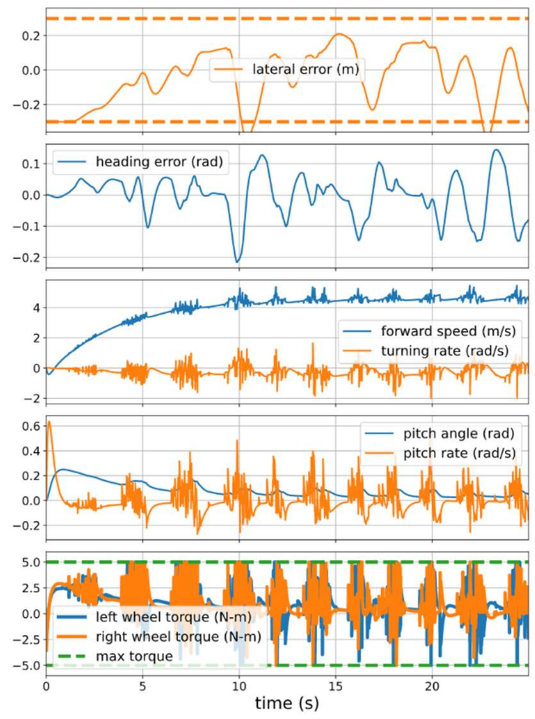

Figure 6: Results of $v_{des} = 5\frac{m}{s}$

The "jitter" and fluctuations increase, as can be clearly seen in the increase in lateral error and heading error. This again is a very interesting consequence of the fact that our track is not a straight line but a circle. When we go fast, the turning needs to be done quicker and so our controller inputs and states fluctuate more.

With the same reasoning, if we predict the behavior of our system with a lower desired velocity, we will get a smoother input and smoother state variables.

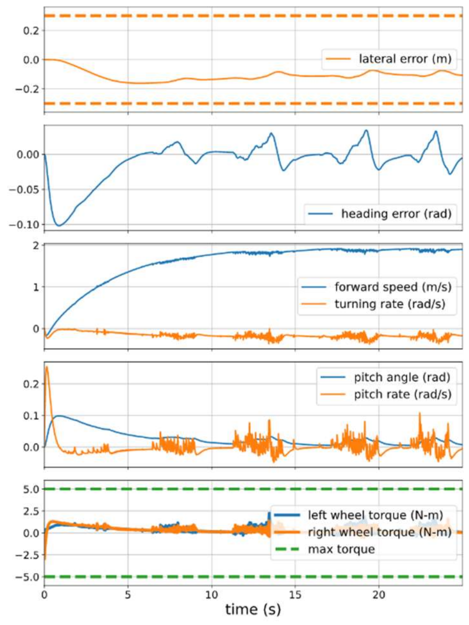

Figure 7: Result of $v_{des} = 2\frac{m}{s}$

The prediction turns out to be true. In fact, the fluctuations are a lot less prominent and the torque inputs are much more reasonable. Therefore, we see that the desired forward velocity, and in general the speed at which our robot is travelling, can drastically affect how the robot's other states are affected. It provides a clear look into the drawbacks of our "straight line" equations of motion and designing a controller around their linearized versions, which is a very interesting problem to try and overcome in future experimentation.

VIII. Conclusion

An LQR feedback controller was designed for the system after the calculated linear system was proven to be controllable. The design of the controller was dictated by the tuning of the cost function (set by Q and R). A custom modification to the simulation that allowed for multiple controllers to be tested at the same time helped get better results faster. Finally, the data shows that the requirements were all met and the verification process was completed successfully. More cases were tested, such as making the robot go at different forward speeds, to understand the underlying assumptions of the designed controller and their effects on the behavior of the final non-linear system.

Acknowledgements

I'd like to gratefully acknowledge the instruction of Professor Timothy Bretl and graduate teaching assistant Jacob Kraft, who provided the guidance and material to successfully complete this project. Their tutorials on the homework and teaching in class formed the basis for the procedure and though process that went into the design of an optimal controller. Their guidance on previous reports was used to improve this one, and their future feedback will help improve this report in the future. I'd also like to acknowledge the help of my peers on the online discussion forum for help with homework and understanding the concepts, with a special mention to Alan Hong for help with determining controllability, Nicholas Hall for his tutorial on modifying urdf files and Quang Do for his guide on adding equation numbers. I'd finally like to acknowledge my own work on Design Project 1, which influenced the way I completed this second design project.

References

There are no additional references for this report. Website references are listed in the footnotes.